Modeling the Sharing of Geographic Data

Richard Pascoe and Neville Churcher

Department of Computer Science
University of Canterbury

Presented at the Second Colloquium of the Spatial Information Research Centre, University of Otago, New Zealand, Nov 28th - Nov 30th 1990.

(Reprinted in the *New Zealand Journal of Computing* January/February 1991)

The recent proliferation of computer networks and the resulting trend toward networked and distributed data base management systems has influenced significantly the sharing of geographical data. The traditional transfer medium of magnetic tape has been replaced by data communication networks and an interfacing strategy has been adopted world-wide which revolves around the use of interchange standards such as SDTS[1], NTF[2], and MACDIF[3]. The functionality and performance of contemporary geographical information systems will be enhanced through their utilization of the increased connectivity of computers. In this paper we develop a method for modeling the data transfer scenarios which occur in practice. We stress the importance of semantic information and show that its inclusion in a machine readable description of the syntax and semantics of transfer file formats allows practical solutions to be developed for problems that may be anticipated during the modeling process. This information is vital if software is to be generated automatically both for encoding and decoding data to and from the format, and for converting data from one format into another. We describe BQL, a notation for specifying transfer file formats, as an application of our ideas.

Introduction

A typical modern GIS stores a volume of data which would have been regarded as huge only a few years ago. The capture of geographical data is often laborious, particularly where processes such as digitizing are involved. These two factors alone are a powerful motivation for exchanging data wherever possible rather than duplicating its capture. In addition, there are often significant political or commercial factors involved. For example, autonomous local government bodies might reasonably expect their GIS to give them access to data held by their peers. The potential for such exchanges is often a major factor in the decision to establish a GIS, yet results in practice are often disappointing.

The traditional perception of data transfer involves the physical exchange of magnetic tapes between sites. Many other options are currently available, including electronic mail and the file transfer protocol (ftp). More significant is the fact that the sharing of data may be achieved

---

[1]Spatial Data Transfer Specification (USA and to be adopted in a modified form as a NZ and as an Australian standard)

[2]National Transfer Standard (Britain)

[3]Map And Chart Data Interchange Format (Canada)

automatically, and often transparently, by software acting either at the operating system level or at the application level. Examples of this include: the network file system (NFS), and networked and distributed database management systems (DBMS).

The recent proliferation of computer networks and the resulting trend towards networked and distributed DBMS dramatically widens the scope for data sharing between geographical information systems. Since a GIS is, in effect, a specialized type of DBMS we should aim to employ both theoretical techniques and practical expertise gained from other areas of database research.

The remainder of the paper is structured as follows. We begin by outlining a framework for categorizing data transfers in terms of a number of interconnected parameters. We briefly discuss the rôle of standardization and suggest that in the long term it may be more appropriate to standardize the means of describing the syntax and semantics of formats, rather than the formats themselves.

In the next section we develop a representation of the data transfer process in terms of database concepts, using the case of relational databases since they are the most relevant to GIS and to data transfers. The potential difficulties arising from an inadequate description of database semantics are outlined in the following section together with our suggested solutions.

A prototype system for the management of data transfers has been developed by one of us (R. P.). We illustrate a component of this system, BQL, by considering the example of a "mini-GINA" format. The means by which further semantic information may be incorporated into the BQL notation are discussed.

Two appendices give, respectively, examples of the difficulties that can arise when insufficient semantic information is available and the details of BQL .

Describing Individual Data Transfer Scenarios

Traditionally, there was only one feasible type of data transfer. Data files from the source GIS were placed onto magnetic tape and transferred onto the computer system of the target GIS. These data files were then processed by an interface, software specifically constructed for converting data from one file format to another, to produce data files in the format expected by the target GIS. Data from these files were then read into the target GIS and integrated with the existing data sets.

As outlined in the introduction, recent advances in data communications enable users to transfer data from one computer system to another via communication networks rather than by magnetic tape through the postal service. Much of the effort of the transfer may be handled by software and the required degree of user involvement may vary. Furthermore, the user may have transparent access to data files whose physical location is hidden by a networked or distributed file system. In the foreseeable future, users with appropriate authorization may reasonably expect to access data on one computer system while using a GIS which is running on another computer system.

To increase our understanding of the diverse data transfer scenarios that are currently achievable, we are developing a technique, analogous to that used to describe queueing systems, for describing individual data transfer scenarios.

A data transfer scenario is described by assigning appropriate values to a set of pre-defined parameters which characterise the fundamental aspects of a data transfer. These parameters

provide a framework for the discussions in later sections of this paper and are mentioned here to indicate our general methodology. The selection of parameters includes:

(1) *Physical transfer parameters.* These are typically inter-related, and include the data transfer medium (mag. tape/ LAN/ WAN…), the degree of direct user involvement and the mode (batch/interactive) of the transfer.

(2) *Source and target data models.* Most DBMS are based on the hierarchical, network or relational data models. Other data models, such as object-oriented, deductive or temporal, are becoming important. A heterogeneous network may link DBMS based on more than one data model. Transfers are simplest when both the source and target GIS use the same internal model of data. For simplicity, the remainder of this paper assumes the relational model of data for both the source and target data models.

(3) *Data model schema comparison index.* Even though the underlying model of data is the same, individual databases may have widely differing schemas. The comparison index indicates how well the tables and fields of the source and target systems correspond. For example, co-ordinate systems may be different or a concept may have different names in the two systems.

(4) *Granularity of data transfer.* If the entire contents of a GIS are involved in the transfer then the acceptable time and resource consumption parameters for the transfer will be considerably different from those for an on-line transfer involving a single polygon.

Having identified the main parameters of a data transfer, we are in a position to determine the best way to take advantage of existing knowledge wherever possible. A considerable body of results is available in some areas while others have received very little attention. We believe that it is important to understand the interaction of the parameters of the transfer. Consider the relationship between the physical transfer parameters and the schema comparison index.

The ease with which data may be physically transferred from one computer system to another may be attributed to the development of international standards for Open System Interconnection (OSI). Fundamental to these standards is a seven level data communication model [Tanenbaum (1981)]. Any standardization of the communication of geographic data should be consistent with the OSI model. Furthermore, existing OSI model standards should be utilized wherever possible. Examples of such standards include: Implementation method (ISO 8211) of the SDTS; Abstract Syntax Notation (ISO 8824). The latter has been used to define MACDIF [Evangalatos *et al* (1989)]; however, the notation may not convey sufficient semantic information to enable automatic transfer to and from MACDIF in all cases.

Increasingly, the physical transfer of data is no longer the barrier which prevents the sharing of data, rather, it is the problem of data translation which is the stumbling block. The data must be coerced, sometimes forcibly, into the format of the target GIS. The optimal techniques for restructuring the data depend on both the schema comparison index and the physical transfer parameters. Data translation occurs as the second of a three phase model of an interface proposed by Pascoe and Penny (1990). By applying results obtained from research into schema integration to the problem of comparing and integrating relational data models we aim to automate the restructuring process.

Relational View of Data Transfers

The relational model of data [Codd (1970)] is currently the most widespread, and many GIS systems are built on top of a relational DBMS. Furthermore, the practical advantages, [Penny (1986), van Roessel and Fosnight (1985), van Roessel et al (1986), Pascoe and Penny (1990)] of using relational operators to transform geographic data from one structure to another are well known. For these reasons, the relational data model is assumed in this paper. The arguments may be readily extended to other data models.

We may describe a relational database as $D[r_i(R_i, \Delta_i)]$. It consists of a set of relations r_i, each having its own schema, R_i, and set Δ_i of dependencies. Queries are then expressed in terms of the operators of the relational algebra (π, σ, $\otimes$, $\cap$, —, $\cup$, $\div$, $\bowtie$, …) or, equivalently, as predicates of the relational calculus. Further details are given in standard texts such as those by Date (1990), Ullman (1982) or Maier (1983).

There are several possible interpretations of the transfer process within the relational model. The following discussion is intended to be illustrative rather than rigorous.

Source and target databases could be envisaged as both being views [Date (1990)] of a single underlying database, which would correspond to a relational form of a standard interchange format. The view definition of each GIS would then be the set of queries required to transfer data from the underlying database. One problem with this interpretation is that there must actually *be* an underlying "standard" database, with all the usual difficulties of managing the evolution of the standard. Thus there is no real advantage over the current equivalent in terms of standard transfer file formats.

We advocate a two-step picture of the transfer process. Within this interpretation, each GIS is prepared to export (make public) and populate a view of its own schema. The exported view may include only part of the entire GIS schema, so that confidential data or connections may be readily preserved. Queries on the view may be used to export data i.e. to populate an external, possibly intermediary, database. In order to transfer data between two GIS it is necessary to develop queries which move data from the relations of one view to those of the other.

We may express transfer operations as operators, ψ, constructed from the operators of the relational algebra. We can consider ψ as acting on both the intension and extension—i.e. involving operations like append as well as project. The complete transfer from initial state D to final state D' is then represented by an operator, Ψ, which is a composition of transformations

$$D'[r_i'(R_i', \Delta_i')] = \Psi(D[r_i(R_i, \Delta_i)]). \quad (1)$$

$$\Psi = \psi^n \psi^{n-1} \ldots \psi^1. \quad (2)$$

The problem is then to construct Ψ, using our knowledge of the semantics of D and D', so that it transforms the database from one valid database state to another. A valid database state is one where every attribute's value is legal and every semantic constraint is obeyed.

This requires us to have a procedure for comparing the two schemas to determine their compatibility and construct semantically valid queries to effect the data translation. Such a procedure should be highly automated wherever possible, implying that the exported schemas should contain as much metadata (self-description) as is possible. This is a special case of schema

integration (sometimes called view integration). Schema integration involves using knowledge of the properties, relationships and constraints of the data items to adjust the schema of a database (GIS) to allow incorporation of new data or new interpretations of existing data. This is a difficult problem for arbitrary databases, and difficulties remain despite considerable research effort [Batini et al. (1987), Biskup & Räsch (1988), Larson et al. (1989)]. However, we believe that the specialized nature of GIS databases will allow effective heuristics to be developed, and manual integration strategies for GIS have already been considered [Nyerges (1989)].

The Rôle of Semantic Information

One of the most important advantages of a relational database is also one of its greatest potential drawbacks. The database typically appears to the user as a set of tables with labelled columns and unordered rows—the relational data model is almost completely devoid of *semantic* information. Semantic information describes, *inter alia*, the purpose and construction of each relation together with the intended connections to be made with other relations.

The relevance of semantic information becomes apparent when we consider queries, since the operations used to create query results are precisely those which would be used to create data for export or to import new data. What effects do traumatic schema modifications, such as those involved in the transfer of data between sites, have on the semantic structure of both exporting and importing databases?

A variety of unpleasant effects may occur if semantic information, such as details of the relationships between relations or attributes within relations, is not available. While many legal sequences of relational operations may be performed on a given database, not all will preserve the database semantics. In formal terms, problems may be characterised as "update dependencies" or as violations of the "dependency preservation" or "lossless join" properties. In conceptual model terms we speak of "connection traps" and "semantic disintegrity" [Howe (1983), Bowers (1988)]. Appendix 1 briefly illustrates the potential impact of semantic problems on a GIS.

In order to determine whether or not a particular query is semantically valid it is necessary to know a great deal about the semantic structure of the database. Where is semantic information kept? This information, sometimes referred to as "metadata," may be stored in a variety of forms:

- In many cases, it is stored only in the minds of the users. This is a particularly unsatisfactory situation, but is very common.

- Some separate form of documentation may be used. Examples include entity-relationship diagrams. Separate documentation has many faults. Some metadata reporting tools are available, but these often require more knowledge of the database schema than it is reasonable to expect the average user to have and are unlikely to be included in a commercially available GIS.

- The DBMS may have mechanisms for imposing specific integrity constraints. Frequently these are restricted to simple constraints—in the case of Ingres [Stonebraker et al. (1976)] they are simple predicates involving constant values, comparison operators and attributes from a single relation. Such simple facilities cannot even enforce constraints such as the referential integrity condition which is a fundamental part of the relational model itself.

- Formal techniques exist to describe the semantics of a database in terms of data dependencies. The most widely used of these are functional dependencies, multi-valued dependencies and join dependencies.

We argue that the final of these possibilities is the most generally applicable to transfer scenarios. Metadata concerning relationships and type hierarchies should also be included. Local "experts" and separate documentation are not appropriate for automated data transfer software. Constraint mechanisms vary between different DBMS and a specification of the constraints is necessary in any case.

The semantic problems discussed above are "before and after" problems—the exact details of the transfer mechanism are not involved. We can view the transfer/integration process as a transformation from one valid database state to another.

Equations (1) and (2) represent the transformation as a composition of operators. A full description of the exported views of the source and target GIS allow us to picture Ψ as

$$\Psi = \Psi_{tj}\Psi_{ji}\Psi_{is} \qquad (3)$$

Where Ψ_{is} is the known sequence of operations defining the exported schema of the source GIS, Ψ_{tj} defines the exported schema of the target GIS and Ψ_{ji} is to be determined using a schema integration procedure.

The relational algebra allows us to re-order some operations and a knowledge of the dependencies allows us to determine potentially troublesome transformations. Once a suitable set of transformations has been determined the data transfer can proceed. The actual transfer may be *implemented* in terms of quite different operations such as file export, but the effect is that of the above composition.

Even relatively simple data transfers may introduce serious semantic problems (see Appendix 1). One might argue that "common sense" would prevent such problems from occurring in practice. However, appeals to common sense are effectively claims that sufficient metadata is known to those performing the transfers. Such arguments are not tenable in the case of automated data transfers, such as we are considering in this work.

In practice, we must be prepared to place reasonable limits on the amount of metadata that is transferred and on the form of its expression. The criteria used should include consideration of the applicability of the information to automated schema integration. Some "reasonable" expectations might include:

- Keys & dependencies. A key is a set of attributes which functionally determines every attribute in the relation schema. There may be several candidate keys for a given relation. A knowledge of the keys of the tables being transferred can tell us a great deal about the dependency structure of the relations. If we assume that relations are in at least 2NF (the key, the whole key…) then only the other "significant" dependencies need be specified explicitly.

- Source relations. The schema and dependencies which hold in the source relations are important if we are to detect violations of the lossless-join and dependency-preservation properties.

- Relation schemas. These are normally supplied anyway, but information on bounds and units of data items is not generally available.

The BQL format specification is capable of extension to include this information (see Appendix 2.). The format schema section currently supplies the relation schemas and primary keys. The inclusion of "significant" dependencies would be straightforward, as would the specification of the relations from which the data elements are drawn. The latter allows us to check for potential semantic problems when considering the target GIS schema.

Rather than include this information as part of each transfer, it may be appropriate for two GIS databases to perform some "schema matching" before regular transfers begin, in order to identify potential difficulties. This corresponds to the determination of the appropriate transformations.

Automated data transfer systems may need to be able to perform semantic integrity checks "on the fly," or to use semantic information to guide their choices between alternative transfer mechanisms. Expert system technology may be appropriate, and the system will maintain a knowledge base derived from previous transfers.

BQL: A Notation for Specifying Data File Formats

Use of the BQL notation for describing data formats [Pascoe (1990)] combines the use of relational database technology and compiler-generating tools. Specifically, Ingres [Stonebraker *et al.* (1976)], yacc [Johnson (1979)], and lex [Lesk and Schmidt (1979)] are used to process data files described by BQL specifications. Given a BQL specification of a transfer file format, a program can be generated (fig. 1) for reading (decoding) data from, and writing (encoding) data to, files of that format. It is the specification of the file format, rather than its contents, that is important for the automatic generation of encoders and decoders.

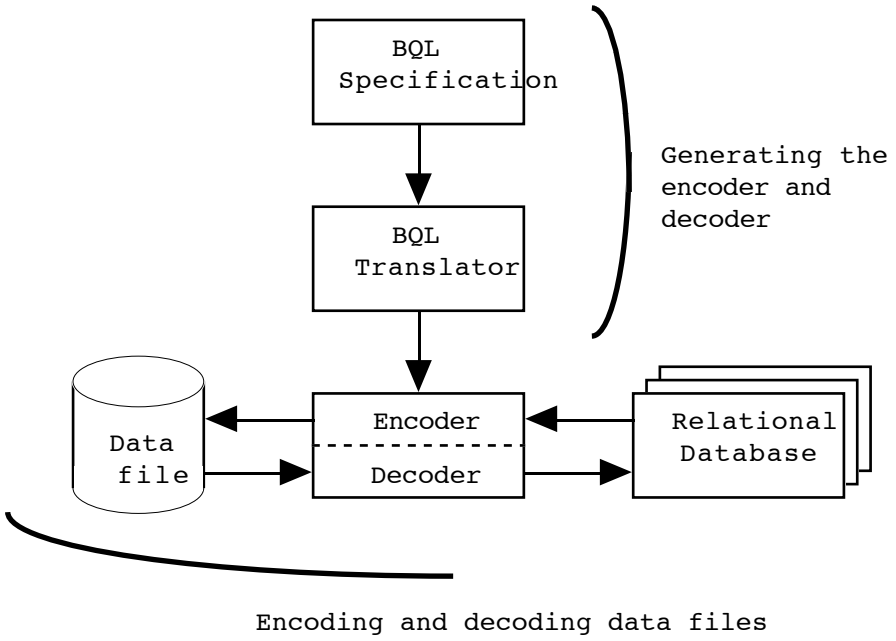

Figure 1: Translation of a BQL specification into an executable program.

A BQL specification of a format consists of four sections. An example format appears in Appendix 2, and further details are given in Pascoe (1990). A definition section is used for maintaining information such as version numbers. The format data dictionary defines the data elements of a format. The dictionary entries define the properties of a data element, specifying the type of the corresponding database attribute together with input and output formats. The format schema defines a set of relations suitable for storing data elements of the format, in a relational data base.

The format implementation method specifies the way in which data elements are stored in a data file. Both the format schema and the format implementation method specification refer to data elements which are defined in the data dictionary.

Conclusion

We have demonstrated the practicality of flexible automated data transfers. Such automated transfers will become increasingly important as GIS takes advantage of advances in computer communications. In order to achieve automated transfers it is necessary to have a formalized means of describing the source and target data formats. These may or may not be in terms of traditional transfer file structures. BQL is capable of representing the semantic information which is required if data is to be successfully translated from one schema to another, manually or automatically. Ultimately, sophisticated systems for sharing geographic data should be developed, based upon current and future standards for both data communications and for notations to be used to describe the diverse structures of geographic data.

Acknowledgement

The authors wish to acknowledge the influence of John Penny on the present work.

References

Batini, C., Lenzerini, M. & Navathe, S. "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Comp. Surv.*, **18**(4), pp323-364, (1987).

Biskup, J. & Räsch, U. "The Equivalence Problem for Relational Database Schemas," in *Proc. Symposium on Mathematical Foundations of Database Systems*, J. Biskup et al. eds, Lecture Notes in Computer Science Vol 305, Springer-Verlag, (1988).

Bowers, D. *From Data to Database*, Van Nostrand Reinhold (UK), (1988).

Codd. E., "A relational model of data for large shared data banks," *Comm. ACM*, **13**(6), pp377-387, (1970).

Date, C. J. *An introduction to Database Systems*, Vol 1 5ed, Addison-Wesley, (1990).

Evangalatos, T., Jiwani , Z., Mckellar, D., O'Brien C.D., "The Telecommunication of Map and Chart Data" , *Proc. Ninth International Symposium on Computer-Assisted Cartography* (AUTO CARTO 9), Baltimore, Maryland, (1989).

Howe, D. *Data Analysis for Database Design*, Arnold, (1983).

Johnson, S. C., Yacc: Yet Another Compiler-Compiler. *Unix Time-Sharing System: Unix Programmer's Manual*, Seventh Edition, Vol.2B (1979).

Larson, J. A. "A Theory of Attribute Equivalence in Databases with Application to Schema Integration," *IEEE Trans. Soft. Eng.*, **15**(4), pp449-463, (1989).

Lesk, M. E. & E. Schmidt, Lex - A Lexical Analyzer Generator. *Unix Time-Sharing System: Unix Programmer's Manual*, Seventh Edition, Volume 2B,(1979).

Maier, D. *The Theory of Relational Databases*, Computer Science Press, (1983).

Nyerges, T., "Schema integration analysis for the development of GIS databases," *Int. J. Geog. Inf. Syst.*, **3**(2), pp153-183, (1989).

Pascoe, R. T, *BQL User Manual*, Technical Report (Draft), Computer Science Department, University of Canterbury (1990).

Pascoe, R. T., & Penny, J. P., "Construction of interfaces for the exchange of geographic data", *Int. J. Geog. Inf. Syst.*, **4**(2), pp147-156, (1990).

Pascoe, R. T., Data translation between geographic information systems. M. Sc. Thesis, Department of Computer Science, University of Canterbury, New Zealand, (1989).

Penny J. P., Relational methods for format conversion of map data. *Cartography*, **15**, pp26-34, (1986).

Stonebraker, M., Wong, E., Kreps, P. & Held, G. "The Design and Implementation of Ingres," *ACM TODS*, **1**(3), pp189-222, (1976).

Tanenbaum, A. S. *Computer Network,s* Prentice/Hall International editions, (1981).

Ullman, J. *Principles of Database Systems*, 2ed, Computer Science Press, (1982).

van Roessel, J. W. and Fosnight, E. A. A relational Approach to vector data structure conversion. *Proc. Auto-carto 7 Conf.* Washington, (1985).

van Roessel, J., Bankers, D., Connochioli, V., Doescher, S., Fosnight G., Wehde, M. & Taylor, D., *Vector Data Structure Conversion at the EROS Data Center*. Final Report (Draft), Phase I, EROS Data Center, (1986).

Appendix 1

Example A

Consider a database containing information about cities (population, area, …), streets (name, length, surface, …) and houses (construction, area, value, …). This information could be structured in the relations:

cities(city#, population, area, …)

streets(streetname, length, surface, …)

houses (house#, construction, area, value, …)

There is no way that this structure can be used to answer apparently reasonable queries like "which street is this house in?" or "how many houses are in this street?" Each tuple of the house relation contains a foreign key value which connects it to the corresponding tuple of the city tuple, and streets and cities are connected similarly. This structure can only satisfy queries like "which streets are in the same city as this house?" or "how many houses are in the same city as this street?" This problem is called a "fan trap" [Howe (1983)].

The structure shown in the relations below is probably a better representation of the semantics of this model.

cities(city#, population, area, …)
streets(streetname, city#, length, surface, …)
houses (house#, streetname, construction, area, value, …)

It shows the essentially hierarchical structure of the relationships between the three entities. However, there may still be difficulties. For example, how should one deal with a house which is not in a city, or one which is not in a street? It may be necessary to construct an artificial street, or to implement the additional relationship by adding an attribute "city#" to the `houses` relation. Such problems, caused by unsupported generation-skipping relationships, are known as "chasm traps" [Howe (1983)].

Example B

Consider a GIS database, controlled by a (hypothetical) Ministry of Agriculture and Fisheries, which includes a relation called `fish_locations`, which stores data about the species of fish found in stretches of rivers with particular flow rates. Some sample data is shown below.

fish_locations

river_name	flow	species
cam	1	carp
amazon	1	piranha
rakaia	5	salmon
dart	5	trout

Now suppose that the (equally hypothetical) Acclimatisation Society wishes to incorporate this data into its own GIS. Since there is an existing relation, river_flows, which holds the data about flow rates in different stretches of rivers, the Society decides to use it and to create a new relation, fish_habitats, to hold the additional information. The river_flows and fish_locations relations would then be filled to give the following state:

fish_habitats

flow	species
1	carp
1	piranha
5	salmon
5	trout

river_flows

river	flow
cam	1
amazon	1
rakaia	5
dart	5

These relations may subsequently be re-joined, perhaps to include additions made by the Society, and transferred to update the Ministry's original fish_locations relation. However, it would not be long before somebody would decide that there was something fishy going on. Some enquiries would reveal that the "new" fish_locations relation now looks like this:

fish_locations´

	river_name	flow	species
	cam	1	carp
+	cam	1	piranha
+	amazon	1	carp
	amazon	1	piranha
	rakaia	5	salmon
+	rakaia	5	trout
+	dart	5	salmon
	dart	5	trout

Spurious tuples (denoted by + above) appear in the relation reconstructed from its two projections. Clearly such tuples violate our data integrity requirements. This effect is known as a lossy join [Ullman (1982)]. Note that the effect is present even when no changes are made to the fish_habitats and river_flows relations before they are re-joined.

In order to have known in advance that this decomposition would cause problems we require some semantic information. In this case we can express this in terms of functional dependencies. The original relation fish_locations ought to be considered as

fish_locations(R, F),

where R = {river, flow, species}, and F = {river → flow, species → flow}.

This states that each river has only one flow rate, and each species is found in water of a single specific flow rate (the truth or realism of this example is not guaranteed!). An elementary result from dependency theory tells us that

$$\pi_{\text{river, flow}}(\textit{fish\_locations}) \bowtie \pi_{\text{species, flow}}(\textit{fish\_locations}) \neq \text{fish\_locations}$$

If the original fish_locations relation had been transferred in its entirety then this problem would not have occurred. However, this is not a practical solution in the general case. A closer examination reveals that if an extra table, river_species(river, species), is included in the transfers and subsequent join then it is possible to recreate the original relation losslessly. In addition, we note that the two relations river_flows and river_species are sufficient to ensure a lossless join.

This may tempt us to omit the fish_habitats relation. To do so would introduce another important semantic problem—that of dependency preservation [Ullman (1982)]. Without fish_habitats the dependency *species* → *flow* is not preserved. Subsequent updates to the two fragments before re-joining could lead to violations of *species* → *flow* in the final relation.

Appendix 2

In the next four sections, a brief description is given of the four parts of a BQL specification.

BQL Definition

Statements such as the RCS compatible version number of the BQL specification are given in this section. For example:

```
$Header$
@=====================================================@
```

The Format Data Dictionary

The dictionary is divided into two sections: the element type definition section, in which a set of types are defined for the data elements in the format; and the data element definition section, in which the data elements themselves are defined using the types given in the first section.

Data element type definitions specify the name of the data element type, the data type of the corresponding attribute in the relational schema, the regular expression used for decoding the data

value from a data file, the printf control string used for encoding the data value into a data file of the format.

The syntactic structure of an element type definition is (using BNF) :

<element type definition> ::=
<element type name> <relational data type> "," <regular expression> ","
<printf control string> ";"

Data element definitions give the name and type of each data item according to the syntax:
<data element definition statement> ::=
<element type name> <list of data element names> ";"
<list of data element names> ::=
<data element name>
| <list of data element names> <data element name>

The Format Schema

The format schema defines the relations in which values of the format's data elements are stored. These relations are defined using the data elements given in the format data dictionary. The syntax for defining a relation is:

<definition of a relation> ::= <relation name> "(" <list of attributes> ")"
<list of attributes> ::= <attribute> | <list of attributes> "," <attribute>
<attribute> ::= <data element name> | "#" <data element name>

Preceeding the name of a data element with the character "#" indicates that the data element is part of the key for this relation. This is an example of the inclusion of semantic information in a specification of a transfer format.

The Format Implementation Method Specification

The basic unit of an implementation method specification is a*symbol* , of which there are three types:*String symbol*s, consisting of any sequence of characters enclosed in quotes. Examples are "Header", "\n" (the newline character) and "Object\n";*Data symbols,* referring to data dictionary elements. Given the relation rln1 with an attribute attr1, an example of a data string is 'rln1.attr1'; and a *Non-terminal symbol,* where the symbol is defined by a BQL production elsewhere in the specification.

A file format is described by a set of BQL productions. A BQL*production* consists of two parts: to the left of a colon is the *production name,* which is a non-terminal symbol that uniquely identifies the production; and to the right is the *production definition,* which is a series of symbols which define the production. The most austere form of a production is one whose definition consists entirely of string symbols. For example:

file_header : "Geographic Data Transfer file UoC format 112" ;

The production name is 'file_header' and the production definition consists of the string symbol ' "Geographic Data Transfer file UoC format 112" '. The semi-colon is used for punctuation.

A production definition may use other productions by including their names in the definition. These production names are substitutes for their definitions: thus, the two BQL productions:

file: header ;
header : “Geographic Data Transfer file UoC\n” ;

are equivalent to the single BQL production:

file : “Geographic Data Transfer file UoC\n” ;

Enclosing the definition of a production in braces, ‘{’ and ‘}’, indicates that the production definition, referred to as a *repetitive production definition*, may be repeated many times. When a production definition includes a data symbol the definition must be enclosed by braces. The following example BQL specification is for the mini-GINA format described in Pascoe [1989].

```
/*=====================================================================*
 *                                                                     *
 *                      Example BQL Specification                      *
 *                     (1 May 1990, $version 1.0)                      *
 *                                                                     *
 *   This BQL specification defines the mini GINA file format as       *
 *   described in Pascoe [1989].                                       *
 *                                                                     */
@=====================================================================@
/*                      The Format Data Dictionary                     *
 *                                                                     *
 *     An entry in the dictionary consists of:                         *
 *     type_name    data type for attribute, regular expression,       *
 *                  printf control string                              */

#define digit [0-9]
#define sign  [+-]
#define E     sign digit digit
#define char  [\,\-A-Za-z'$#!%^&~`:?/_]
#define space [\t ]

types
       integer smallint,sign ? digit +,“%d” ;
       real    float,
               (sign ? digit + ”.” digit * ([E] sign digit digit)*)
           |   (sign ? digit * ”.” digit + ([E] sign digit digit)*),
               “%lf” ;
       string  varchar(255),
               \“( char digit * space *) * \” | (char digit *) +,
               “%s” ;

elements
       integer fid, layer, network flw seq;
       string fc, feattype, crdtype;
       real gp1, gp2, x, y, z;
@=====================================================================@
/*                          The Format Schema                          *
 *                                                                     *
 *     An entry in the schema consists of:                             *
 *                                                                     *
 *     relation_name  (  [#]data element, [#]data element, .. )        *
 *                                                                     *
 *     The # symbol indicates that the attribute is (part of) the      *
 *     the key.The reserved attribute named ‘seq’ will be used to      *
 *     explicitly sequence a list of items.                            */
```

```
       dosfeat ( #fid, fc, layer, network, feattype, crdtype, gp1,
                 gp2, flw )
       doscrds ( #fid, #seq, x, y, z)

@=====================================================================@
/*          The Format Implementation Method Specification          */

datafile :
       "UDB-FEAT\n" { feature_dfn(dosfeat.fid)
                      { crd_dfn(dosfeat.fid) } } ;

feature_dfn(dosfeat.fid) :
       "feat " dosfeat.fid dosfeat.fc dosfeat.layer dosfeat.network
               dosfeat.feattype ;

crd_dfn(dosfeat.fid) :
       "coor " { doscrds.x doscrds.y " " } "\n" ;
```